PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F	A2	(11) International Publication Number: WO 00/34844
		(43) International Publication Date: 15 June 2000 (15.06.00)

(21) International Application Number: PCT/US99/28782

(22) International Filing Date: 7 December 1999 (07.12.99)

(30) Priority Data:
09/208,741 8 December 1998 (08.12.98) US

(63) Related by Continuation (CON) or Continuation-in-Part (CIP) to Earlier Application
US 09/208,741 (CIP)
Filed on 8 December 1998 (08.12.98)

(71) Applicant *(for all designated States except US)*: JEDI TECHNOLOGIES, INC. [US/US]; 2200 Laurelwood Road, Santa Clara, CA 95054 (US).

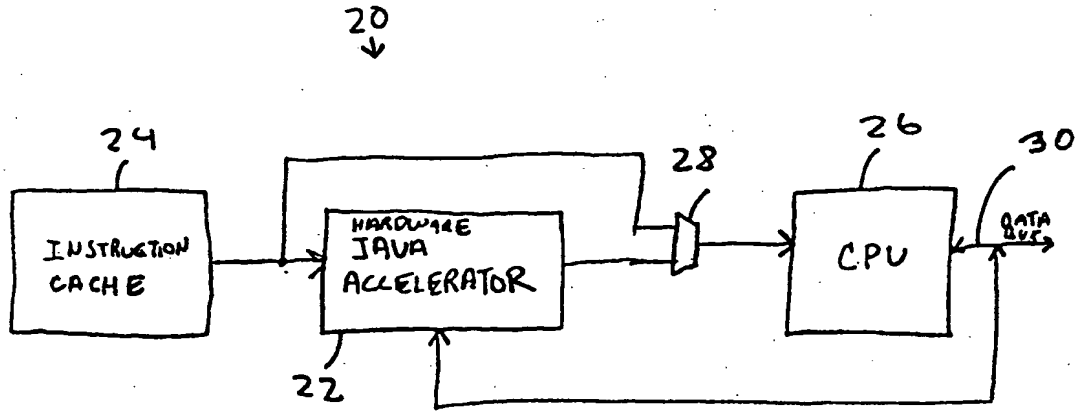
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: PATEL, Mukesh, K. [US/US]; 787 Boar Circle, Fremont, CA 94539 (US). KAMDAR, Jay [US/US]; 22455 Palm Avenue, Cupertino, CA 95014 (US). RANGANTH, V., R. [US/US]; 1043 Sandalwood Lane, Milpitas, CA 95035 (US).

(74) Agent: KREBS, Robert, E.; Burns, Doane, Swecker & Mathis, LLP, P.O. Box 1404, Alexandria, VA 22313–1404 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: JAVA VIRTUAL MACHINE HARDWARE FOR RISC AND CISC PROCESSORS



(57) Abstract

A hardware Java accelerator is provided to implement portions of the Java virtual machine in hardware in order to accelerate the operation of the system on Java bytecodes. The Java hardware accelerator preferably includes Java bytecode translation into native CPU instructions. The combination of the Java hardware accelerator and a CPU provides an embedded solution which results in an inexpensive system to run Java programs for use in commercial appliances.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

EPSDOCID: <WO __0034844A2_I_>

JAVA VIRTUAL MACHINE HARDWARE FOR RISC AND CISC PROCESSORS

BACKGROUND OF THE INVENTION

Java ™ is an object orientated programming language developed by Sun Microsystems. The Java language is small, simple and portable across platforms and operating systems, both at the source and at the binary level. This makes the Java programming language very popular on the Internet.

Java's platform independence and code compaction are the most significant advantages of Java over conventional programming languages. In conventional programming languages, the source code of a program is sent to a compiler which translates the program into machine code or processor instructions. The processor instructions are native to the system's processor. If the code is compiled on an Intel-based system, the resulting program will only run on other Intel-based systems. If it is desired to run the program on another system, the user must go back to the original source code, obtain a compiler for the new processor, and recompile the program into the machine code specific to that other processor.

Java operates differently. The Java compiler takes a Java program and, instead of generating machine code for a particular processor, generates bytecodes. Bytecodes are instructions that look like machine code, but aren't specific to any processor. To execute a Java program, a bytecode interpreter takes the Java bytecode converts them to equivalent native processor instructions and executes the Java program. The Java byte code interpreter is one component of the Java Virtual Machine.

Having the Java programs in bytecode form means that instead of being specific to any one system, the programs can run on any platform and any operating system as long a Java Virtual Machine is available. This allows a binary bytecode file to be executable across platforms.

The disadvantage of using bytecodes is execution speed. System specific programs that run directly on the hardware from which they are compiled, run significantly faster that Java bytecodes, which must be processed by the Java Virtual Machine. The processor must both convert the Java bytecodes into native instructions in the Java Virtual Machine and execute the native instructions.

One way to speed up the Java Virtual Machine is by techniques such as the "Just in Time" (JIT) interpreter, and even faster interpreters known as "Hot Spot JITs" interpreters. The JIT versions all result in a JIT compile overhead to generate native processor instructions. These JIT interpreters also result in additional memory overhead.

The slow execution speed of Java and overhead of JIT interpreters have made it difficult for consumer appliances requiring local-cost solutions with minimal memory usage and low energy consumption to run Java programs. The performance requirements for existing processors using the fastest JITs more than double to support running the Java Virtual Machine in software. The processor performance requirements could be met by employing superscalar processor architectures or by increasing the processor clock frequency. In both cases, the power requirements are dramatically increased. The memory bloat that results from JIT techniques, also goes against the consumer application requirements of low cost and low power.

It is desired to have an improved system for implementing Java programs that provides a low-cost solution for running Java programs for consumer appliances.

SUMMARY OF THE INVENTION

The present invention generally relates to a Java hardware accelerator which can be used to quickly translate Java bytecodes into native instructions for a central processing unit (CPU). The hardware accelerator speeds up the processing of the Java bytecodes significantly because it removes the bottleneck which previously occurred

when the Java Virtual Machine is run in software on the CPU to translate Java bytecodes into native instructions.

In the present invention, at least part of the Java Virtual Machine is implemented in hardware as the Java hardware accelerator. The Java hardware accelerator and the CPU can be put together on a single semiconductor chip to provide an embedded system appropriate for use with commercial appliances. Such an embedded system solution is less expensive than a powerful superscalar CPU and has a relatively low power consumption.

The hardware Java accelerator can convert the stack-based Java bytecodes into a register-based native instructions on a CPU. The hardware accelerators of the present invention are not limited for use with Java language and can be used with any stack-based language that is to be converted to register-based native instructions. Also, the present invention can be used with any language that uses instructions, such as bytecodes, which run on a virtual machine.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention may be further understood from the following description in conjunction with the drawings.

Figure 1 is a diagram of the system of the present invention including the hardware Java accelerator.

Figure 2 is a diagram illustrating the use of the hardware Java accelerator of the present invention.

Figure 3 is a diagram illustrating some the details of a Java hardware accelerator of one embodiment of the present invention.

Figure 4 is a diagram illustrating the details of one embodiment of a Java accelerator instruction translation in the system of the present invention.

Figure 5 is a diagram illustration the instruction translation operation of one embodiment of the present invention.

Figure 6 is a diagram illustrating the instruction translation system of one embodiment of the present invention using instruction level parallelism.

Figures 7A-7D are a table showing one possible list of bytecodes which can cause exceptions in a preferred embodiment

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figure 1 is a diagram of the system 20 showing the use of a hardware Java accelerator 22 in conjunction with a central processing unit 26. The Java hardware accelerator 22 allows part of the Java Virtual Machine to be implemented in hardware. This hardware implementation speeds up the processing of the Java byte codes. In particular, in a preferred embodiment, the translation of the Java bytecodes into native processor instructions is at least partially done in the hardware Java accelerator 22. This translation has been part of a bottleneck in the Java Virtual Machine when implemented in software. In Figure 1, instructions from the instruction cache 24 or other memory is supplied to the hardware Java accelerator 22. If these instruction are Java bytecode, the hardware Java accelerator 22 can convert these bytecodes into native processor instruction which are supplied through the multiplexer 28 to the CPU. If a non-Java code is used, the hardware accelerator can be by-passed using the multiplexer 26.

The Java hardware accelerator can do, some or all of the following tasks:

1. Java bytecode decode;
2. identifying and encoding instruction level parallelism (ILP), wherever possible;
3. translating bytecodes to native instructions;
4. managing the Java stack on a register file associated with the CPU or as a separate stack;

5. generating exceptions on instructions on predetermined Java byte codes;
6. switching to native CPU operation when native CPU code is provided;
7. performing bounds checking on array instructions; and
8. managing the variables on the register file associated with the CPU.

In a preferred embodiment, the Java Virtual Machine functions of bytecode interpreter, Java register, and Java stack are implemented in the hardware Java accelerator. The garbage collection heap and constant pool area can be maintained in normal memory and accessed through normal memory referencing.

The major advantages of the Java hardware accelerator is to increase the speed in which the Java Virtual Machine operates, and allow existing native language legacy applications, software base, and development tools to be used. A dedicated microprocessor in which the Java bytecodes were the native instructions would not have accesss to those legacy applications.

Although the Java hardware accelerator is shown in Figure 1 as separate from the central processing unit, the Java hardware accelerator can be incorporated into a central processing unit. In that case, the central processing unit has a Java hardware accelerator subunit to translate Java bytecode into the native instructions operated on by the main portion of the CPU.

Figure 2 is a state machine diagram that shows the operation of one embodiment of the present invention. Block 32 is the power-on state. During power-on, the multiplexer 28 is set to bypass the Java hardware accelerator. In block 34, the native instruction boot-up sequence is run. Block 36 shows the system in the native mode executing native instructions and by-passing the Java hardware accelerator.

In block 38, the system switches to the Java hardware accelerator mode. In the Java hardware accelerator mode, Java bytecode is transferred to the Java hardware accelerator 22, converted into native instructions then sent to the CPU for operation.

The Java accelerator mode can produce exceptions at certain Java bytecodes. These bytecodes are not processed by the hardware accelerator 22 but are processed in

the CPU 26. As shown in block 40, the system operates in the native mode but the Java Virtual Machine is implemented in the CPU which does the bytecode translation and handles the exception created in the Java accelerator mode.

The longer and more complicated bytecodes that are difficult to handle in hardware can be selected to produce the exceptions. Figure 7 is a table showing one possible list of bytecodes which can cause exceptions in a preferred embodiment.

Figure 3 is a diagram illustrating details of one embodiment of the Java hardware accelerator of the present invention. The Java hardware accelerator includes Java accelerator instruction translation hardware 42. The instruction translation Unit 42 is used to convert Java bytecodes to native instructions. One embodiment of the Java accelerator instruction translation hardware 42 is described in more detail below with respect to Figure 4. This instruction translation hardware 42 uses data stored in hardware Java registers 44. The hardware Java Registers store the Java Registers defined in the Java Virtual Machine. The Java Registers contain the state of the Java Virtual Machine, affect its operation, and are updated after each bytecode is executed. The Java registers in the Java virtual machine include the PC, the program counter indicating what bytecode is being executed; Optop, a pointer to the top of the operand stack; Frame, a pointer to the execution environment of the current method; and Vars, a pointer to the first local variable available of the currently executing method. The virtual machine defines these registers to be a single 32-bit word wide. The Java registers are also stored in the Java stack which can be implemented as the hardware Java stack 50 or the Java stack can be stored into the CPU associated register file.

In a preferred embodiment, the hardware Java registers 44 can include additional registers for the use of the instruction translation hardware 42. These registers can include a register indicating a switch to native instructions and a register indicating the version number of the system.

The Java PC can be used to obtain bytecode instructions from the instruction cache 24. In one embodiment the Java PC is multiplexed with the normal program counter 54 of the central processing unit 26 in multiplexer 52. The normal PC 54 is

not used during the operation of the Java hardware bytecode translation. In another embodiment, the normal program counter 54 is used as the Java program counter.

The Java registers are a part of the Java Virtual Machine and should not be confused with the general registers 46 or 48 which are operated upon by the central processing unit 26. In one embodiment, the system uses the traditional CPU register file 46 as well as a Java CPU register file 48. When native code is being operated upon the multiplexer 56 connects the conventional register file 46 to the execution logic 26c of the CPU 26. When the Java hardware accelerator is active, the Java CPU register file 48 substitutes for the conventional CPU register file 46. In another embodiment, the conventional CPU register file 46 is used.

As described below with respect to Figures 3 and 4, the Java CPU register file 48, or in an alternate embodiment the conventional CPU register file 46, can be used to store portions of the operand stack and some of the variables. In this way, the native register-based instructions from the Java accelerator instruction translator 42 can operate upon the operand stack and variable values stored in the Java CPU register file 48, or the values stored in the conventional CPU register file 46. Data can be written in and out of the Java CPU register file 48 from the data cache or other memory 58 through the overflow/underflow line 60 connected to the memory arbiter 62. The overflow/underflow transfer of data to and from the memory to can done concurrently with the CPU operation. Alternately, the overflow/underflow transfer can be done explicitly while the CPU is not operating. The overflow/underflow bus 60 can be implemented as a tri-state bus or as two separate buses to read data in and write data out of the register file when the Java stack overflows or underflows.

The register files for the CPU could alternately be implemented as a single register file with native instructions used to manipulate the loading of operand stack and variable values to and from memory. Alternately, multiple Java CPU register files could be used: one register file for variable values, another register file for the operand stack values, and another register file for the Java frame stack holding the method environment information.

The Java accelerator controller (co-processing unit) 64 can be used to control the hardware Java accelerator, read in and out from the hardware Java registers 44 and Java stack 50, and flush the Java accelerator instruction translation pipeline upon a "branch taken" signal from the CPU execute logic 26c.

The CPU 26 is divided into pipeline stages including the instruction fetch 26a, instruction decode 26b, execute logic 26c, memory access logic 26d, and writeback logic 26e. The execute logic 26c executes the native instructions and thus can determine whether a branch instruction is taken and issue the "branch taken" signal.

Figure 4 illustrates an embodiment of a Java accelerator instruction translator which can be used with the present invention. The instruction buffer 70 stores the bytecode instructions from the instruction cache. The bytecodes are sent to a parallel decode unit 72 which decodes multiple bytecodes at the same time. Multiple bytecodes are processed concurrently in order to allow for instruction level parallelism. That is, multiple bytecodes may be converted into a lesser number of native instructions.

The decoded bytecodes are sent to a state machine unit 74 and Arithmetic Logic Unit (ALU) 76. The ALU 76 is provided to rearrange the bytecode instructions to make them easier to be operated on by the state machine 74. The state machine 74 converts the bytecodes into native instructions using the look-up table 78. Thus, the state machine 74 provides an address which indicates the location of the desired native instruction in the look-up table 78 . Counters are maintained to keep a count of how many entries have been placed on the operand stack, as well as to keep track of the top of the operand stack. In a preferred embodiment, the output of the look-up table 78 is augmented with indications of the registers to be operated on at line 80. The register indications are from the counters and interpreted from bytecodes. Alternately, these register indications can be sent directly to the Java CPU register file 48 shown in Figure 3.

The state machine 74 has access to the Java registers in 44 as well as an indication of the arrangement of the stack and variables in the Java CPU register file

48 or in the conventional CPU register file 46. The buffer 82 supplies the translated native instructions to the CPU.

The operation of the Java hardware accelerator of one embodiment of the present invention is illustrated in Figures 5 and 6. Figure 5, section I shows the instruction translation of the Java bytecode. The Java bytecode corresponding to the mnemonic *iadd* is interpreted by the Java virtual machine as an integer operation taking the top two values of the operand stack, adding them together and pushing the result on top of the operand stack. The Java translating machine translates the Java bytecode into a native instruction such as the instruction ADD R1, R2. This is an instruction native to the CPU indicating the adding of value in register R1 to the value in register R2 and the storing of this result in register R2 . R1 and R2 are the top two entries in the operand stack.

As shown in Figure 5, section II, the Java register includes a PC value of "Value A" that is incremented to "Value A+1". The Optop value changes from "Value B" to "Value B-1" to indicate that the top of the operand stack is at a new location. The Vars value which points to the top of the variable list is not modified. In Figure 5, section III, the contents of a Java CPU register file, such as the Java CPU register file 48 in Figure 3, is shown. The Java CPU register file starts off with registers R0-R5 containing operand stack values and registers R6-R7 containing variable values. Before the operation of the native instruction, register R1 contains the top value of the operand stack. Register R6 contains the first variable. After the execution of the native instruction, register R2 now contains the top value of the operand stack. Register R1 no longer contains a valid operand stack value and is available to be overwritten by a operand stack value from the memory sent across the overflow/underflow line 60 or from the bytecode stream.

Figure 5, section IV shows the memory locations of the operand stack and variables which can be stored in the data cache 58 or in main memory. For convenience, the memory is illustrated without illustrating any virtual memory scheme. Before the native instruction executes, the address of the top of the operand

stack, Optop, is "Value B". After the native instruction executes, the address of the top of the operand stack is "Value B-1" containing the result of the native instruction. Note that the operand stack value "4427" can be written into register R1 across the overflow/underflow line 60. Upon a switch back to the native mode, the data in the Java CPU register file 48 should be written to the data memory.

Consistency must be maintained between the Hardware Java Registers 44, the Java CPU register file 48 and the data memory. The CPU 26 and Java Accelerator Instruction Translation Unit 42 are pipelined and any changes to the hardware java registers 44 and changes to the control information for the Java CPU register file 48 must be able to be undone upon a "branch taken" signal. The system preferably uses buffers (not shown) to ensure this consistency. Additionally, the Java instruction translation must be done so as to avoid pipeline hazards in the instruction translation unit and CPU.

Figure 6 is a diagram illustrating the operation of instruction level parallelism with the present invention. In Figure 6 the Java bytecodes *iload_*n and *iadd* are converted by the Java bytecode translator to the single native instruction ADD R6, R1. In the Java Virtual Machine, *iload_n* pushes the top local variable indicated by the by the Java register VAR onto the operand stack.

In the present invention the Java hardware translator can combine the *iload_n* and *iadd* bytecode into a single native instruction. As shown in figure 6, section II, the Java Register, PC, is updated from "Value A" to "Value A+2". The Optop value remains "value B". The value Var remains at "value C".

As shown in Figure 6, section III, after the native instruction ADD R6, R1 executes the value of the first local variable stored in register R6, "1221", is added to the value of the top of the operand stack contained in register R1 and the result stored in register R1. In Figure 6, section IV, the Optop value does not change but the value in the top of the register contains the result of the ADD instruction, 1371.

The Java hardware accelerator of the present invention is particularly well suited to a embedded solution in which the hardware accelerator is positioned on the

same chip as the existing CPU design. This allows the prior existing software base and development tools for legacy applications to be used. In addition, the architecture of the present embodiment is scalable to fit a variety of applications ranging from smart cards to desktop solutions. This scalability is implemented in the Java accelerator instruction translation unit of Figure 4. For example, the lookup table 78 and state machine 74 can be modified for a variety of different CPU architectures. These CPU architectures include reduced instruction set computer (RISC) architectures as well as complex instruction set computer (CISC) architectures. The present invention can also be used with superscalar CPUs or very long instruction word (VLIW) computers.

While the present invention has been described with reference to the above embodiments, this description of the preferred embodiments and methods is not meant to be construed in a limiting sense. For example, the term Java in the specification or claims should be construed to cover successor programming languages or other programming languages using basic Java concepts (the use of generic instructions, such as bytecodes, to indicate the operation of a virtual machine). It should also be understood that all aspects of the present invention are not to be limited to the specific descriptions, or to configurations set forth herein. Some modifications in form and detail the various embodiments of the disclosed invention, as well as other variations in the present invention, will be apparent to a person skilled in the art upon reference to the present disclosure. It is therefore contemplated that the following claims will cover any such modifications or variations of the described embodiment as falling within the true spirit and scope of the present invention.

CLAIMS

1. A system comprising:

a central processing unit; and

a hardware accelerator operably connected to the central processing unit, the hardware accelerator adapted to translate stack-based instructions into register-based instructions native to the central processing unit.

2. The system of Claim 1, wherein the stack-based instructions are associated with a virtual machine.

3. The system of Claim 1, wherein the stack-based instructions are Java bytecode.

4. The system of Claim 1, wherein the hardware accelerator implements at least part of a Java virtual machine.

5. The system of Claim 1, wherein the hardware accelerator is connected between a memory and the central processing unit.

6. The system of Claim 5, wherein the hardware accelerator is connected between an instruction cache and the central processing unit.

7. The system of Claim 1, wherein the hardware accelerator is adapted to manage a java stack.

8. The system of Claim 1, wherein the hardware accelerator is adapted to store at least some of a Java operand stack in a register file connected to the central processing unit.

9. The system of Claim 8, wherein the hardware accelerator has access to the data bus of the central processing unit.

10. The system of Claim 8, wherein the hardware accelerator is adapted to swap parts of the operand stack are in and out of the register file from a memory.

11. The system of Claim 8, wherein the central processing unit is operably connected to a native register file and a register file controlled by the hardware accelerator

12. The system of Claim 11, wherein the at least some of the Java operand stack is stored in the register file controlled by the hardware accelerator.

13. The system of Claim 8, wherein the central processing unit is operably connected to a native register file and wherein the at least some of the Java operand stack is stored in the native register file.

14. The system of Claim 8, wherein the hardware controller is further adapted to store at least some variables in the register file.

15. The system of Claim 8, wherein the hardware accelerator is incorporated within the central processing unit.

16. The system of Claim 1, wherein the hardware accelerator has access to at least one bus of the central processing unit.

17. The system of Claim 1, wherein the hardware accelerator is adapted to examine the stack-based instructions to determine whether multiple stack-based instructions can be combined into fewer register-based instructions.

18. The system of Claim 17, wherein multiple stack-based instructions pass through the hardware accelerator concurrently to allow for the operation of the combining logic.

19. The system of Claim 1, wherein the hardware accelerator is divided into pipelined stages.

20. The system of Claim 1, wherein the hardware accelerator is adapted to be flushed under predetermined conditions.

21. The system of Claim 1, wherein the central processing unit and hardware accelerator are on the same chip.

22. The system of Claim 1, wherein the hardware accelerator produces an exception upon at least one of the stack-based instructions, and wherein the central processing unit will, in software, translate the at least one of the stack-based instructions causing the exception.

23. The system of Claim 1, wherein the hardware accelerator is incorporated within the central processing unit.

24. A system comprising:

a central processing unit; and

a hardware java accelerator operably connected to the central processing unit, the hardware java accelerator adapted to translate java bytecodes into instructions native to the central processing unit.

25. A system comprising:

a central processing unit; and

a hardware accelerator operably connected to the central processing unit, the hardware accelerator adapted implement at least part of a virtual machine associated with a computer language, the hardware accelerator adapted to translate instructions for the virtual machine into native instructions for the central processing unit.

26 . A method comprising:

moving a stack-based instruction from memory to a hardware accelerator;

in the hardware accelerator, converting the stack-based instruction into a register-based instruction native to a central processing unit; and

in the central processing unit, executing the register-based instruction.

27. The method of Claim 26, wherein the stack-based instructions are associated with a virtual machine.

28. The method of Claim 26, wherein the stack-based instructions are Java bytecode.

29. The method of Claim 26, wherein the accelerator implements at least part of a Java virtual machine.

30. The method of Claim 26, further comprising, in the hardware accelerator, managing a java stack.

31. The method of Claim 26, further comprising storing at least some of a Java operand stack in a register file connected to the central processing unit.

32. The method of Claim 26, wherein parts of the operand stack are swapped in and out of the register file from a memory by the hardware accelerator.

33. The method of Claim 26, wherein the hardware accelerator examines the stack-based instructions to determine whether multiple stack-based instructions can be combined into fewer register-based instructions.

34. The method of Claim 26, further comprising producing an exception in the hardware accelerator upon at least one stack-based instruction, and translating the at least one stack-based instruction causing the exception in software in the central processing unit.

35. The method of Claim 26, wherein the central processing unit and hardware accelerator are on the same chip.

36. The method of Claim 26, wherein the hardware accelerator is incorporated within the central processing unit.

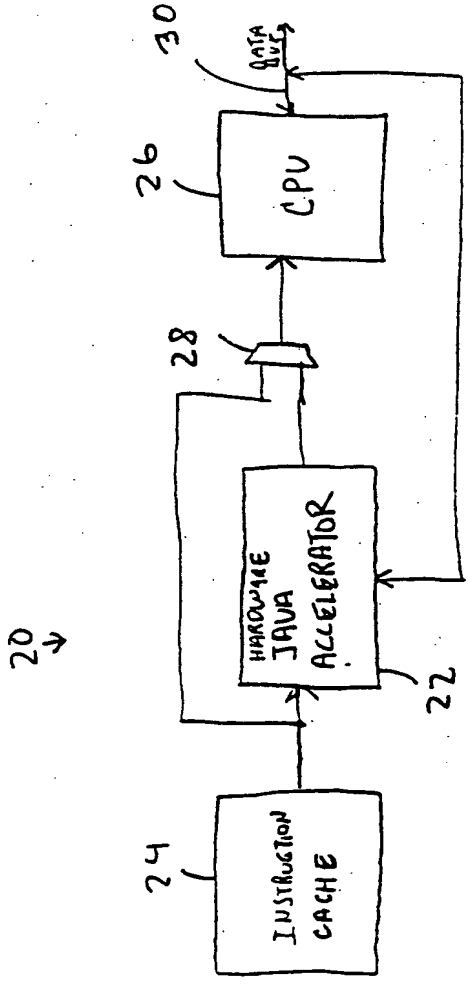


FIGURE 1

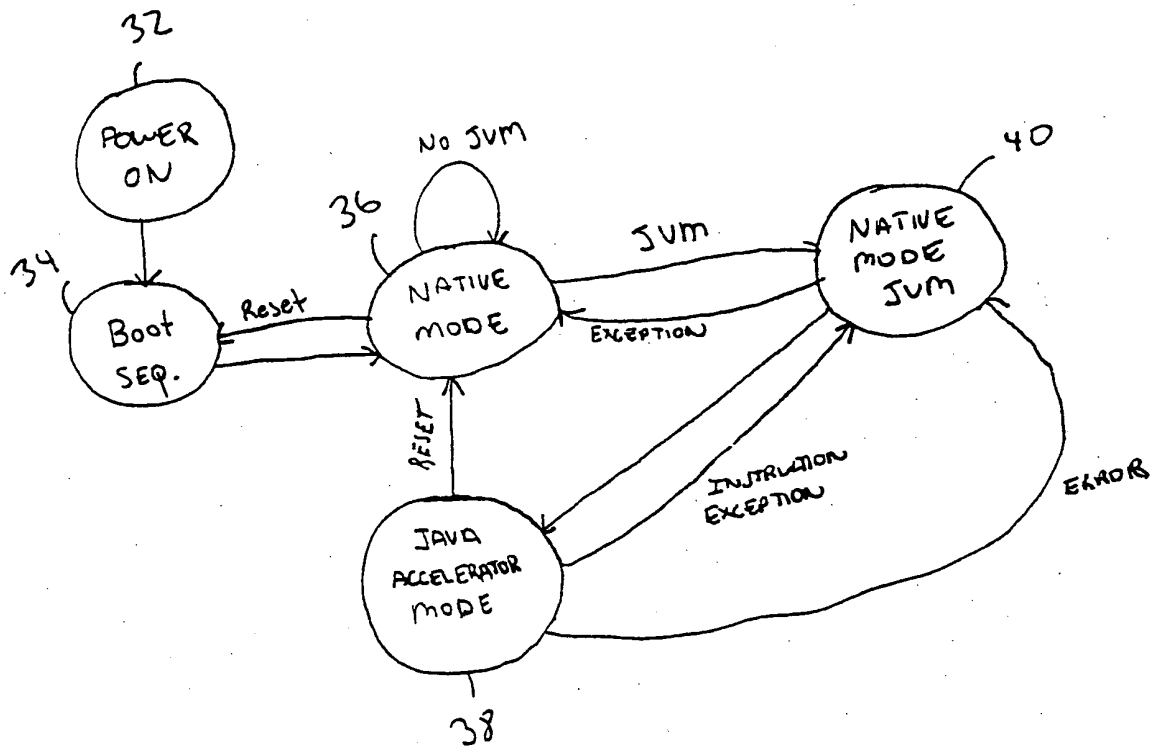

32
POWER ON
34
Boot SEQ.
Reset
36
NATIVE MODE
No JVM
JVM
EXCEPTION
40
NATIVE MODE JVM
RESET
INSTRUCTION EXCEPTION
ERROR
JAVA ACCELERATOR MODE
38

FIGURE 2

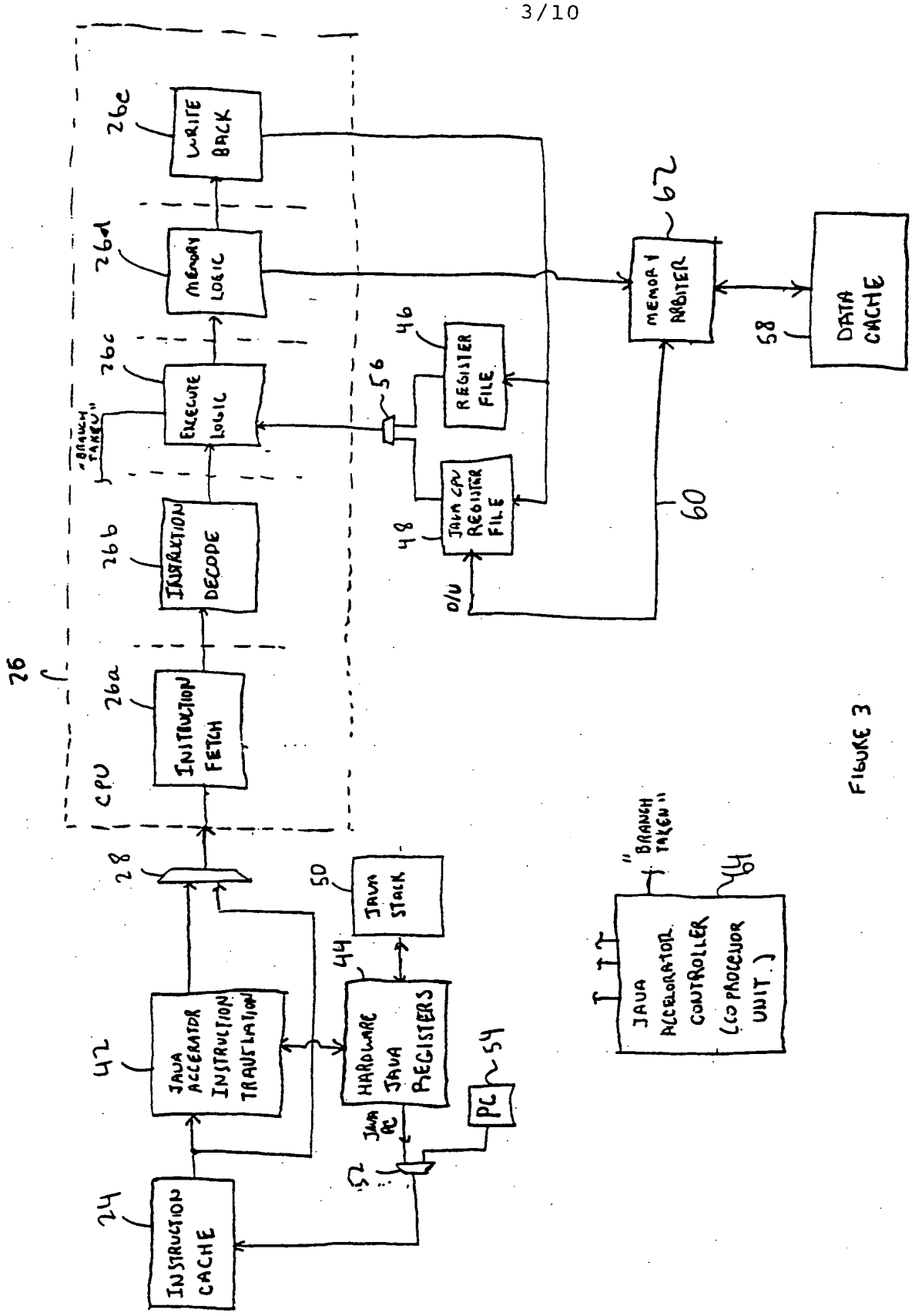


FIGURE 3

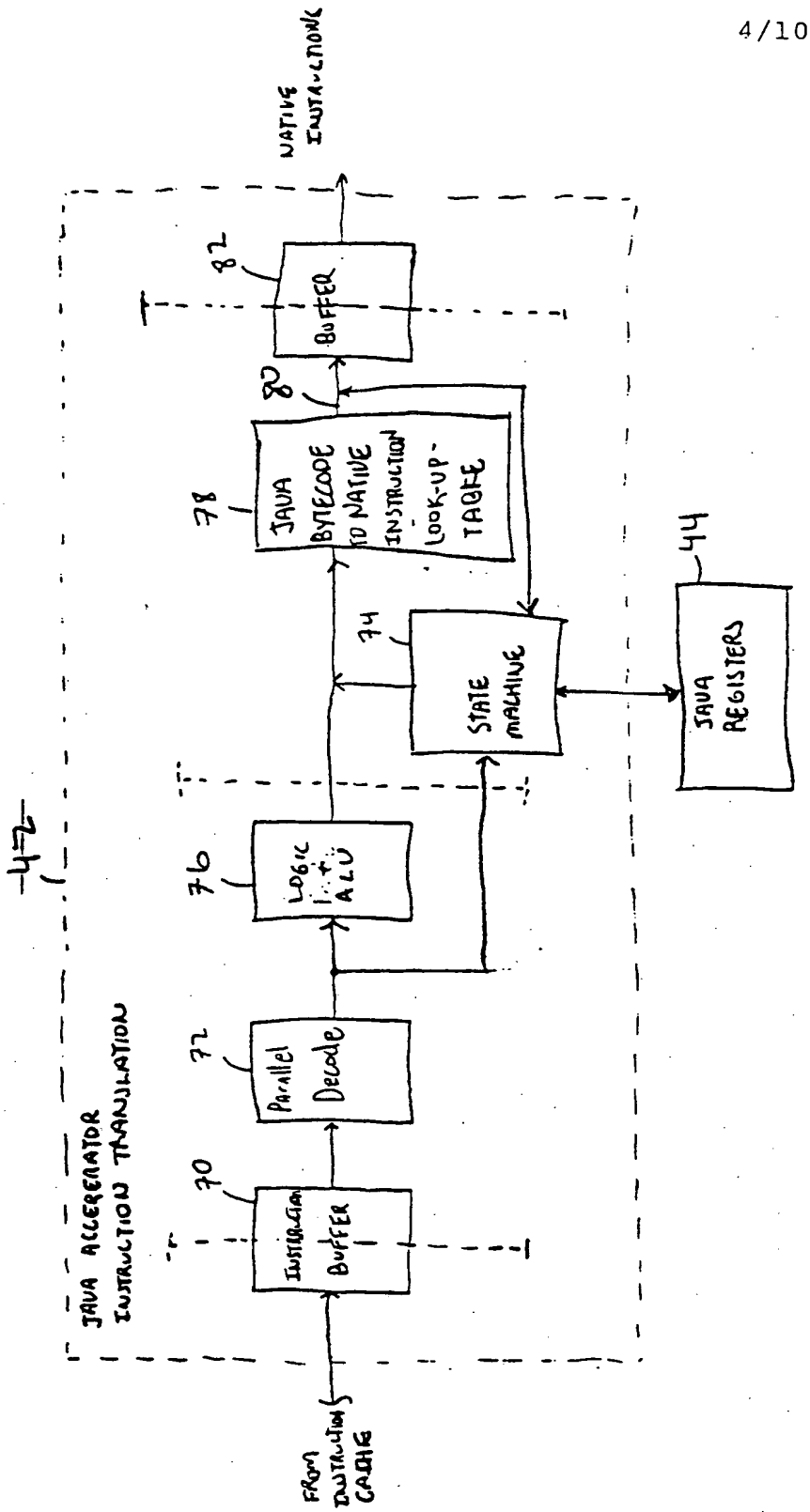

NATIVE INSTRUCTIONS
82
BUFFER
80
78
JAVA BYTECODE TO NATIVE INSTRUCTION LOOK-UP-TABLE
74
STATE MACHINE
44
JAVA REGISTERS
76
LOGIC ALU
72
Parallel Decode
70
INSTRUCTION BUFFER
JAVA ACCELERATOR INSTRUCTION TRANSLATION
FROM INSTRUCTION CACHE

FIGURE 4

I. INSTRUCTION TRANSLATION

JAVA BYTECODE	⇒	NATIVE INSTRUCTION
iadd		ADD R1, R2

II. JAVA REGISTER

PC = VALUE A	⇒	PC = VALUE A + 1
OPTOP = VALUE B (R1)		OPTOP = VALUE B - 1 (R2)
VAR = VALUE C		VAR = VALUE C

III. JAVA CPU REGISTER FILE

	R0	0001		R0	0001
contains value of top of operand stack →	R1	0150	Not a valid stack value →	R1	0150
	R2	1210	contains →	R2	1360
	R3	0007	value of the top of operand stack	R3	0007
	R4	0005		R4	0005
	R5	0006		R5	0006
contains first variable →	R6	1221		R6	1221
	R7	1361		R7	1361

IV. MEMORY

PTOP =	VALUE B ⇒	0150			0150
	(VALUE B-1)	1210	OPTOP = VALUE B-1		1360
		0007			0007
		0005			0005
		0006			0006
		0001			0001
		4427			4427
VAR =	VALUE C	1221	VAR = VALUE C		1221
		1361			1361
		1101			1101

FIGURE 5

I. INSTRUCTION TRANSLATION

JAVA BYTECODE		NATIVE INSTRUCTION
iload_n iadd	⇒	ADD R6, R1

II. JAVA REGISTER

PC = VALUE A OPTOP = VALUE B (R1) VAR = VALUE C	⇒	PC = VALUE A + 2 OPTOP = VALUE B (R1) VAR = VALUE C

III. JAVA CPU REGISTER FILE

		R0	0001		R0 0001
contains value of top of operand stack	→	R1	0150	⇒ contains value of top of stack	→ R1 1371
		R2	1210		R2 1210
		R3	0007		R3 0007
		R4	0005		R4 0005
		R5	0006		R5 0006
contain first variable	→	R6	1221	Contains first variable	→ R6 1221
		R7	1361		R7 1361

IV. MEMORY

OPTOP = VALUE B	-	0150
	-	1210
	-	0007
	-	0005
	-	0006
	-	0001
	-	4427
VAR = VALUE C	-	1221
	-	1361
	-	1101

OPTOP = VALUE B	-	1371
	-	1210
	-	0007
	-	0005
	-	0006
	-	0001
	-	4427
VAR = VALUE C	-	1221
	-	1361
	-	1101

FIGURE 6

Opcodes Mnemonic	Opcode xHH	Excep Gen
nop	0x00	
aconst_null	x01	
iconst_m1	x02	
iconst_n(0-5)	x03 - x08	
lconst_n(0-1)	x09 - x0a	
fconst_n(0-2)	x0c - x0d	
dconst_n(0-1)	x0e -x0f	
bipush	x10	
sipush	x11	
ldc	x12	y
ldc_w	x13	y
ldc2_w	x14	y
iload	x15	
lload	x16	
fload	x17	
dload	x18	
aload	x19	
iload_n(0-3)	x1a - x1d	
lload_n(0-3)	x1e - x21	
fload_n(0-3)	x22 - x25	
dload_n(0-3)	x26 - x29	
aload_n(0-3)	x2a - x2d	
iaload	x2e	
laload	x2f	
faload	x30	
daload	x31	
aaload	x32	
baload	x33	
caload	x34	
saload	x35	
istore	x36	
lstore	x37	
fstore	x38	
dstroe	x39	
astroe	x3a	
istore_n(0-3)	x3b - x3e	
lstore_n(0-3)	x3f - x42	
fstore_n(0-3)	x43 - x46	
dstore_n(0-3)	x47 - x4a	
astore_n(0-3)	x4b - x4e	
iastore	x4f	
lastore	x50	
fastroe	x51	
dastore	x52	
bastore	x53	
aastore	x54	
castroe	x55	
sastore	x56	

FIGURE 7A

pop	x57	
pop2	x58	
dup	x59	
dup_x1	x5a	
dup_x2	x5b	
dup2	x5c	
dup2_x1	x5d	
dup2_x2	x5e	
swap	x5f	
iadd	x60	
ladd	x61	
fadd	x62	y
dadd	x63	y
isub	x64	
lsub	x65	
fsub	x66	y
dsub	x67	y
imul	x68	
lmul	x69	
fmul	x6a	y
dmul	x6b	y
idiv	x6c	y
ldiv	x6d	y
fdiv	x6e	y
ddiv	x6f	y
irem	x70	y
lrem	x71	y
frem	x72	y
drem	x73	y
ineg	x74	
lneg	x75	
fneg	x76	y
dneg	x77	y
ishl	x78	
lshl	x79	
ishr	x7a	
lshr	x7b	
iushr	x7c	
lushr	x7d	
iand	x7e	
land	x7f	
ior	x80	
lor	x81	
ixor	x82	
lxor	x83	
iinc	x84	
i2l	x85	y
i2f	x86	y
i2d	x87	y
l2i	x88	y
l2f	x89	y
l2d	x8a	y

FIGURE 7B

f2i	x8b	y
f2l	x8c	y
f2d	x8d	y
d2i	x8e	y
d2l	x8f	y
d2f	x90	y
i2b	x91	
i2c	x92	
i2s	x93	
lcmp	x94	y
fcmpl	x95	y
fcmpg	x96	y
dcmpl	x97	y
dcmpg	x98	y
ifeq	x99	
ifne	x9a	
iflt	x9b	
ifge	x9c	
ifgt	x9d	
ifle	x9e	
if_icmpeq	x9f	
if_icmpne	xa0	
if_icmplt	xa1	
if_acmpge	xa2	
if_cmpgt	xa3	
if_icmple	xa4	
if_acmpeq	xa5	
if_acmpne	xa6	
goto	xa7	
jsr	xa8	
ret	xa9	
tableswitch	xaa	y
lookupswitch	xab	y
ireturn	xac	
lreturn	xad	
freturn	xae	
dreturn	xaf	
areturn	xb0	
return	xb1	
getstatic	xb2	y
putstatic	xb3	y
getfield	xb4	y
putfield	xb5	y
invokevirtual	xb6	y
invokespecial	xb7	y
invokestatic	xb8	y
invokeinterface	xb9	y
xxunsedxxx	xba	y
new	xbb	y
newarray	xbc	y
anewarray	xbd	y
arraylength	xbe	y

FIGURE 7C

athrow	xbf	y
checkcast	xco	y
instanceof	xc1	y
monitorenter	xc2	y
monitorexit	xc3	y
wide	xc4	y
multianewarray	xc5	y
ifnull	xc6	y
ifnonnull	xc7	y
goto_w	xc8	
jsr_w	xc9	
ldc_quick	xcb	y
ldc_w_quick	xcc	y
ldc2_w_quick	xcd	y
getfield_quick	xce	y
putfield_quick	xcf	y
getfield2_quick	xd0	y
putfield2_quick	xd1	y
getstatic_quick	xd2	y
putstatic_quick	xd3	y
gtestatic2_quick	xd4	y
putstatic2_quick	xd5	y
invokevirtual_quick	xd6	y
invokenonvirtual_quick	xd7	y
invokesuper_quick	xd8	y
invokestatic_quick	xd9	y
invokeinterface_quick	xda	y
invokevirtualobject_quick	xdb	y
new_quick	xdc	y
anewarray_quick	xde	y
multinewarray_quick	xdf	y
checkcast_quick	xe0	y
instanceof_quick	xe1	y
invokevirtual_quick_w	xe2	y
getfield_quick_w	xe3	y
putfield_quick_w	xe4	y
breakpoint	xca	y
impdep1	xfe	y
impdep2	xff	y

FIGURE 7 D

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 9/45, 9/445

A3

(11) International Publication Number: **WO 00/34844**

(43) International Publication Date: 15 June 2000 (15.06.00)

(21) International Application Number: PCT/US99/28782

(22) International Filing Date: 7 December 1999 (07.12.99)

(30) Priority Data:
09/208,741 8 December 1998 (08.12.98) US

(63) Related by Continuation (CON) or Continuation-in-Part (CIP) to Earlier Application
US 09/208,741 (CIP)
Filed on 8 December 1998 (08.12.98)

(71) Applicant *(for all designated States except US)*: JEDI TECHNOLOGIES, INC. [US/US]; 2200 Laurelwood Road, Santa Clara, CA 95054 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: PATEL, Mukesh, K. [US/US]; 787 Boar Circle, Fremont, CA 94539 (US). KAMDAR, Jay [US/US]; 22455 Palm Avenue, Cupertino, CA 95014 (US). RANGANTH, V., R. [US/US]; 1043 Sandalwood Lane, Milpitas, CA 95035 (US).

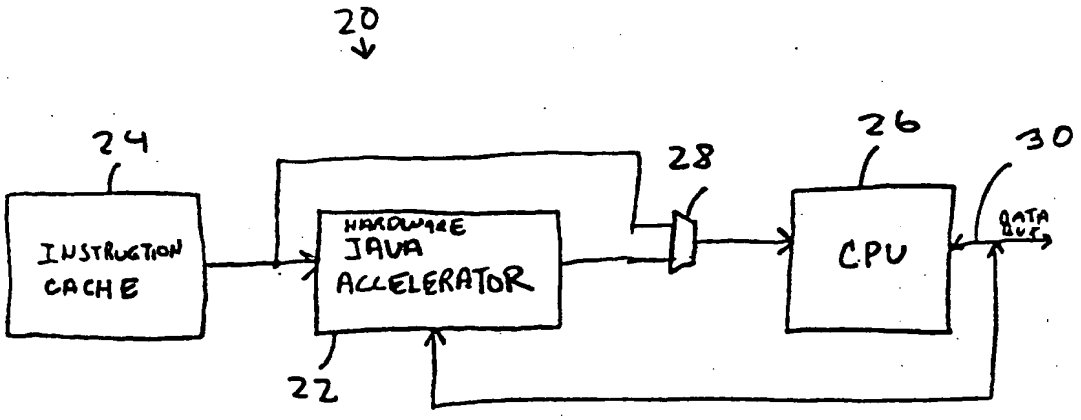
(74) Agent: KREBS, Robert, E.; Burns, Doane, Swecker & Mathis, LLP, P.O. Box 1404, Alexandria, VA 22313–1404 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(88) Date of publication of the international search report:
23 November 2000 (23.11.00)

(54) Title: JAVA VIRTUAL MACHINE HARDWARE FOR RISC AND CISC PROCESSORS



(57) Abstract

A hardware Java accelerator (22) is provided to implement portions of the Java virtual machine in hardware in order to accelerate the operation of the system on Java bytecodes. The Java hardware accelerator (22) preferably includes Java bytecode translation into native CPU instructions. The combination of the Java hardware accelerator and a CPU (26) provides an embedded solution which results in an inexpensive system to run Java programs for use in commercial appliances.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/28782

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 9/45, 9/445

US CL : 717/5

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 717/5, 395/800, 395/570,

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

STN, WEST, EAST, IEEE

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	CACAO- a 64 bit Java VM, just-in-time compiler, November 1997, vol 9, Krall et al, page 1017-1030.	1-36
A, P	US 5,898,885 A (DICKOL et al) 27 April 1999, see entire document	1-36
A, P	US 5,898,850 A (DICKOL et al) 27 April 1999, see entire document	1-36
A, P	US 5,875,336 A (DECKOL et al) 23 February 1999, see entire document	1-36
A	Efficient Java VM Just-in-Time Compilation, IEEE, 1998, see entire document	1-36

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report
21 MARCH 2000	27 JUN 2000

Name and mailing address of the ISA/US	Authorized officer
Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	TARIQ HAFIZ for Regina Fogan
Facsimile No. (703) 305-0040	Telephone No. (703) 305-9643

Form PCT/ISA/210 (second sheet) (July 1998) *

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/28782

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	SGI Webforce 02 is a one-step Web authoring platform, InfoWorld, 20 January 1997, see entire document	1-36

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/28782

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☒ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
15 June 2000 (15.06.2000)

PCT

(10) International Publication Number
WO 00/34844 A3

(51) International Patent Classification[7]: G06F 9/45, 9/445

(21) International Application Number: PCT/US99/28782

(22) International Filing Date: 7 December 1999 (07.12.1999)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/208,741 8 December 1998 (08.12.1998) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 09/208,741 (CIP)
Filed on 8 December 1998 (08.12.1998)

(71) Applicant *(for all designated States except US)*: **JEDI TECHNOLOGIES, INC.** [US/US]; 2200 Laurelwood Road, Santa Clara, CA 95054 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **PATEL, Mukesh, K.** [US/US]; 787 Boar Circle, Fremont, CA 94539 (US). **KAMDAR, Jay** [US/US]; 22455 Palm Avenue, Cupertino, CA 95014 (US). **RANGANTH, V., R.** [US/US]; 1043 Sandalwood Lane, Milpitas, CA 95035 (US).

(74) Agent: **KREBS, Robert, E.**; Burns, Doane, Swecker & Mathis, LLP, P.O. Box 1404, Alexandria, VA 22313-1404 (US).

(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

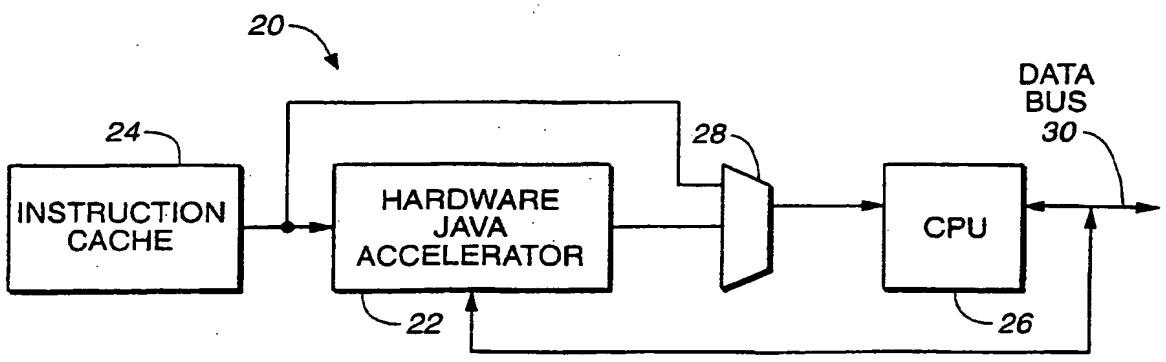
(88) Date of publication of the international search report:
23 November 2000

(48) Date of publication of this corrected version:
25 April 2002

(15) Information about Correction:
see PCT Gazette No. 17/2002 of 25 April 2002, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: JAVA VIRTUAL MACHINE HARDWARE FOR RISC AND CISC PROCESSORS



(57) Abstract: A hardware Java accelerator (22) is provided to implement portions of the Java virtual machine in hardware in order to accelerate the operation of the system on Java bytecodes. The Java hardware accelerator (22) preferably includes Java bytecode translation into native CPU instructions. The combination of the Java hardware accelerator and a CPU (26) provides an embedded solution which results in an inexpensive system to run Java programs for use in commercial appliances.

WO 00/34844 A3

JAVA VIRTUAL MACHINE HARDWARE FOR RISC AND CISC PROCESSORS

BACKGROUND OF THE INVENTION

Java ™ is an object orientated programming language developed by Sun Microsystems. The Java language is small, simple and portable across platforms and operating systems, both at the source and at the binary level. This makes the Java programming language very popular on the Internet.

Java's platform independence and code compaction are the most significant advantages of Java over conventional programming languages. In conventional programming languages, the source code of a program is sent to a compiler which translates the program into machine code or processor instructions. The processor instructions are native to the system's processor. If the code is compiled on an Intel-based system, the resulting program will only run on other Intel-based systems. If it is desired to run the program on another system, the user must go back to the original source code, obtain a compiler for the new processor, and recompile the program into the machine code specific to that other processor.

Java operates differently. The Java compiler takes a Java program and, instead of generating machine code for a particular processor, generates bytecodes. Bytecodes are instructions that look like machine code, but aren't specific to any processor. To execute a Java program, a bytecode interpreter takes the Java bytecode converts them to equivalent native processor instructions and executes the Java program. The Java byte code interpreter is one component of the Java Virtual Machine.

Having the Java programs in bytecode form means that instead of being specific to any one system, the programs can run on any platform and any operating system as long a Java Virtual Machine is available. This allows a binary bytecode file to be executable across platforms.

The disadvantage of using bytecodes is execution speed. System specific programs that run directly on the hardware from which they are compiled, run significantly faster that Java bytecodes, which must be processed by the Java Virtual Machine. The processor must both convert the Java bytecodes into native instructions in the Java Virtual Machine and execute the native instructions.

One way to speed up the Java Virtual Machine is by techniques such as the "Just in Time" (JIT) interpreter, and even faster interpreters known as "Hot Spot JITs" interpreters. The JIT versions all result in a JIT compile overhead to generate native processor instructions. These JIT interpreters also result in additional memory overhead.

The slow execution speed of Java and overhead of JIT interpreters have made it difficult for consumer appliances requiring local-cost solutions with minimal memory usage and low energy consumption to run Java programs. The performance requirements for existing processors using the fastest JITs more than double to support running the Java Virtual Machine in software. The processor performance requirements could be met by employing superscalar processor architectures or by increasing the processor clock frequency. In both cases, the power requirements are dramatically increased. The memory bloat that results from JIT techniques, also goes against the consumer application requirements of low cost and low power.

It is desired to have an improved system for implementing Java programs that provides a low-cost solution for running Java programs for consumer appliances.

SUMMARY OF THE INVENTION

The present invention generally relates to a Java hardware accelerator which can be used to quickly translate Java bytecodes into native instructions for a central processing unit (CPU). The hardware accelerator speeds up the processing of the Java bytecodes significantly because it removes the bottleneck which previously occurred

when the Java Virtual Machine is run in software on the CPU to translate Java bytecodes into native instructions.

In the present invention, at least part of the Java Virtual Machine is implemented in hardware as the Java hardware accelerator. The Java hardware accelerator and the CPU can be put together on a single semiconductor chip to provide an embedded system appropriate for use with commercial appliances. Such an embedded system solution is less expensive than a powerful superscalar CPU and has a relatively low power consumption.

The hardware Java accelerator can convert the stack-based Java bytecodes into a register-based native instructions on a CPU. The hardware accelerators of the present invention are not limited for use with Java language and can be used with any stack-based language that is to be converted to register-based native instructions. Also, the present invention can be used with any language that uses instructions, such as bytecodes, which run on a virtual machine.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention may be further understood from the following description in conjunction with the drawings.

Figure 1 is a diagram of the system of the present invention including the hardware Java accelerator.

Figure 2 is a diagram illustrating the use of the hardware Java accelerator of the present invention.

Figure 3 is a diagram illustrating some the details of a Java hardware accelerator of one embodiment of the present invention.

Figure 4 is a diagram illustrating the details of one embodiment of a Java accelerator instruction translation in the system of the present invention.

Figure 5 is a diagram illustration the instruction translation operation of one embodiment of the present invention.

Figure 6 is a diagram illustrating the instruction translation system of one embodiment of the present invention using instruction level parallelism.

Figures 7A-7D are a table showing one possible list of bytecodes which can cause exceptions in a preferred embodiment

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figure 1 is a diagram of the system 20 showing the use of a hardware Java accelerator 22 in conjunction with a central processing unit 26. The Java hardware accelerator 22 allows part of the Java Virtual Machine to be implemented in hardware. This hardware implementation speeds up the processing of the Java byte codes. In particular, in a preferred embodiment, the translation of the Java bytecodes into native processor instructions is at least partially done in the hardware Java accelerator 22. This translation has been part of a bottleneck in the Java Virtual Machine when implemented in software. In Figure 1, instructions from the instruction cache 24 or other memory is supplied to the hardware Java accelerator 22. If these instruction are Java bytecode, the hardware Java accelerator 22 can convert these bytecodes into native processor instruction which are supplied through the multiplexer 28 to the CPU. If a non-Java code is used, the hardware accelerator can be by-passed using the multiplexer 26.

The Java hardware accelerator can do, some or all of the following tasks:

1. Java bytecode decode;
2. identifying and encoding instruction level parallelism (ILP), wherever possible;
3. translating bytecodes to native instructions;
4. managing the Java stack on a register file associated with the CPU or as a separate stack:

5. generating exceptions on instructions on predetermined Java byte codes;
6. switching to native CPU operation when native CPU code is provided;
7. performing bounds checking on array instructions; and
8. managing the variables on the register file associated with the CPU.

In a preferred embodiment, the Java Virtual Machine functions of bytecode interpreter, Java register, and Java stack are implemented in the hardware Java accelerator. The garbage collection heap and constant pool area can be maintained in normal memory and accessed through normal memory referencing.

The major advantages of the Java hardware accelerator is to increase the speed in which the Java Virtual Machine operates, and allow existing native language legacy applications, software base, and development tools to be used. A dedicated microprocessor in which the Java bytecodes were the native instructions would not have accesss to those legacy applications.

Although the Java hardware accelerator is shown in Figure 1 as separate from the central processing unit, the Java hardware accelerator can be incorporated into a central processing unit. In that case, the central processing unit has a Java hardware accelerator subunit to translate Java bytecode into the native instructions operated on by the main portion of the CPU.

Figure 2 is a state machine diagram that shows the operation of one embodiment of the present invention. Block 32 is the power-on state. During power-on, the multiplexer 28 is set to bypass the Java hardware accelerator. In block 34, the native instruction boot-up sequence is run. Block 36 shows the system in the native mode executing native instructions and by-passing the Java hardware accelerator.

In block 38, the system switches to the Java hardware accelerator mode. In the Java hardware accelerator mode, Java bytecode is transferred to the Java hardware accelerator 22, converted into native instructions then sent to the CPU for operation.

The Java accelerator mode can produce exceptions at certain Java bytecodes. These bytecodes are not processed by the hardware accelerator 22 but are processed in

the CPU 26. As shown in block 40, the system operates in the native mode but the Java Virtual Machine is implemented in the CPU which does the bytecode translation and handles the exception created in the Java accelerator mode.

The longer and more complicated bytecodes that are difficult to handle in hardware can be selected to produce the exceptions. Figure 7 is a table showing one possible list of bytecodes which can cause exceptions in a preferred embodiment.

Figure 3 is a diagram illustrating details of one embodiment of the Java hardware accelerator of the present invention. The Java hardware accelerator includes Java accelerator instruction translation hardware 42. The instruction translation Unit 42 is used to convert Java bytecodes to native instructions. One embodiment of the Java accelerator instruction translation hardware 42 is described in more detail below with respect to Figure 4. This instruction translation hardware 42 uses data stored in hardware Java registers 44. The hardware Java Registers store the Java Registers defined in the Java Virtual Machine. The Java Registers contain the state of the Java Virtual Machine, affect its operation, and are updated after each bytecode is executed. The Java registers in the Java virtual machine include the PC, the program counter indicating what bytecode is being executed; Optop, a pointer to the top of the operand stack; Frame, a pointer to the execution environment of the current method; and Vars, a pointer to the first local variable available of the currently executing method. The virtual machine defines these registers to be a single 32-bit word wide. The Java registers are also stored in the Java stack which can be implemented as the hardware Java stack 50 or the Java stack can be stored into the CPU associated register file.

In a preferred embodiment, the hardware Java registers 44 can include additional registers for the use of the instruction translation hardware 42. These registers can include a register indicating a switch to native instructions and a register indicating the version number of the system.

The Java PC can be used to obtain bytecode instructions from the instruction cache 24. In one embodiment the Java PC is multiplexed with the normal program counter 54 of the central processing unit 26 in multiplexer 52. The normal PC 54 is

not used during the operation of the Java hardware bytecode translation. In another embodiment, the normal program counter 54 is used as the Java program counter.

The Java registers are a part of the Java Virtual Machine and should not be confused with the general registers 46 or 48 which are operated upon by the central processing unit 26. In one embodiment, the system uses the traditional CPU register file 46 as well as a Java CPU register file 48. When native code is being operated upon the multiplexer 56 connects the conventional register file 46 to the execution logic 26c of the CPU 26. When the Java hardware accelerator is active, the Java CPU register file 48 substitutes for the conventional CPU register file 46. In another embodiment, the conventional CPU register file 46 is used.

As described below with respect to Figures 3 and 4, the Java CPU register file 48, or in an alternate embodiment the conventional CPU register file 46, can be used to store portions of the operand stack and some of the variables. In this way, the native register-based instructions from the Java accelerator instruction translator 42 can operate upon the operand stack and variable values stored in the Java CPU register file 48, or the values stored in the conventional CPU register file 46. Data can be written in and out of the Java CPU register file 48 from the data cache or other memory 58 through the overflow/underflow line 60 connected to the memory arbiter 62. The overflow/underflow transfer of data to and from the memory to can done concurrently with the CPU operation. Alternately, the overflow/underflow transfer can be done explicitly while the CPU is not operating. The overflow/underflow bus 60 can be implemented as a tri-state bus or as two separate buses to read data in and write data out of the register file when the Java stack overflows or underflows.

The register files for the CPU could alternately be implemented as a single register file with native instructions used to manipulate the loading of operand stack and variable values to and from memory. Alternately, multiple Java CPU register files could be used: one register file for variable values, another register file for the operand stack values, and another register file for the Java frame stack holding the method environment information.

The Java accelerator controller (co-processing unit) 64 can be used to control the hardware Java accelerator, read in and out from the hardware Java registers 44 and Java stack 50, and flush the Java accelerator instruction translation pipeline upon a "branch taken" signal from the CPU execute logic 26c.

The CPU 26 is divided into pipeline stages including the instruction fetch 26a, instruction decode 26b, execute logic 26c, memory access logic 26d, and writeback logic 26e. The execute logic 26c executes the native instructions and thus can determine whether a branch instruction is taken and issue the "branch taken" signal.

Figure 4 illustrates an embodiment of a Java accelerator instruction translator which can be used with the present invention. The instruction buffer 70 stores the bytecode instructions from the instruction cache. The bytecodes are sent to a parallel decode unit 72 which decodes multiple bytecodes at the same time. Multiple bytecodes are processed concurrently in order to allow for instruction level parallelism. That is, multiple bytecodes may be converted into a lesser number of native instructions.

The decoded bytecodes are sent to a state machine unit 74 and Arithmetic Logic Unit (ALU) 76. The ALU 76 is provided to rearrange the bytecode instructions to make them easier to be operated on by the state machine 74. The state machine 74 converts the bytecodes into native instructions using the look-up table 78. Thus, the state machine 74 provides an address which indicates the location of the desired native instruction in the look-up table 78 . Counters are maintained to keep a count of how many entries have been placed on the operand stack, as well as to keep track of the top of the operand stack. In a preferred embodiment, the output of the look-up table 78 is augmented with indications of the registers to be operated on at line 80. The register indications are from the counters and interpreted from bytecodes. Alternately, these register indications can be sent directly to the Java CPU register file 48 shown in Figure 3.

The state machine 74 has access to the Java registers in 44 as well as an indication of the arrangement of the stack and variables in the Java CPU register file

48 or in the conventional CPU register file 46. The buffer 82 supplies the translated native instructions to the CPU.

The operation of the Java hardware accelerator of one embodiment of the present invention is illustrated in Figures 5 and 6. Figure 5, section I shows the instruction translation of the Java bytecode. The Java bytecode corresponding to the mnemonic *iadd* is interpreted by the Java virtual machine as an integer operation taking the top two values of the operand stack, adding them together and pushing the result on top of the operand stack. The Java translating machine translates the Java bytecode into a native instruction such as the instruction ADD R1, R2. This is an instruction native to the CPU indicating the adding of value in register R1 to the value in register R2 and the storing of this result in register R2 . R1 and R2 are the top two entries in the operand stack.

As shown in Figure 5, section II, the Java register includes a PC value of "Value A" that is incremented to "Value A+1". The Optop value changes from "Value B" to "Value B-1" to indicate that the top of the operand stack is at a new location. The Vars value which points to the top of the variable list is not modified. In Figure 5, section III, the contents of a Java CPU register file, such as the Java CPU register file 48 in Figure 3, is shown. The Java CPU register file starts off with registers R0-R5 containing operand stack values and registers R6-R7 containing variable values. Before the operation of the native instruction, register R1 contains the top value of the operand stack. Register R6 contains the first variable. After the execution of the native instruction, register R2 now contains the top value of the operand stack. Register R1 no longer contains a valid operand stack value and is available to be overwritten by a operand stack value from the memory sent across the overflow/underflow line 60 or from the bytecode stream.

Figure 5, section IV shows the memory locations of the operand stack and variables which can be stored in the data cache 58 or in main memory. For convenience, the memory is illustrated without illustrating any virtual memory scheme. Before the native instruction executes, the address of the top of the operand

stack, Optop, is "Value B". After the native instruction executes, the address of the top of the operand stack is "Value B-1" containing the result of the native instruction. Note that the operand stack value "4427" can be written into register R1 across the overflow/underflow line 60. Upon a switch back to the native mode, the data in the Java CPU register file 48 should be written to the data memory.

Consistency must be maintained between the Hardware Java Registers 44, the Java CPU register file 48 and the data memory. The CPU 26 and Java Accelerator Instruction Translation Unit 42 are pipelined and any changes to the hardware java registers 44 and changes to the control information for the Java CPU register file 48 must be able to be undone upon a "branch taken" signal. The system preferably uses buffers (not shown) to ensure this consistency. Additionally, the Java instruction translation must be done so as to avoid pipeline hazards in the instruction translation unit and CPU.

Figure 6 is a diagram illustrating the operation of instruction level parallelism with the present invention. In Figure 6 the Java bytecodes *iload_*n and *iadd* are converted by the Java bytecode translator to the single native instruction ADD R6, R1. In the Java Virtual Machine, *iload_n* pushes the top local variable indicated by the by the Java register VAR onto the operand stack.

In the present invention the Java hardware translator can combine the *iload_n* and *iadd* bytecode into a single native instruction. As shown in figure 6, section II, the Java Register, PC, is updated from "Value A" to "Value A+2". The Optop value remains "value B". The value Var remains at "value C".

As shown in Figure 6, section III, after the native instruction ADD R6, R1 executes the value of the first local variable stored in register R6, "1221", is added to the value of the top of the operand stack contained in register R1 and the result stored in register R1. In Figure 6, section IV, the Optop value does not change but the value in the top of the register contains the result of the ADD instruction, 1371.

The Java hardware accelerator of the present invention is particularly well suited to a embedded solution in which the hardware accelerator is positioned on the

same chip as the existing CPU design. This allows the prior existing software base and development tools for legacy applications to be used. In addition, the architecture of the present embodiment is scalable to fit a variety of applications ranging from smart cards to desktop solutions. This scalability is implemented in the Java accelerator instruction translation unit of Figure 4. For example, the lookup table 78 and state machine 74 can be modified for a variety of different CPU architectures. These CPU architectures include reduced instruction set computer (RISC) architectures as well as complex instruction set computer (CISC) architectures. The present invention can also be used with superscalar CPUs or very long instruction word (VLIW) computers.

While the present invention has been described with reference to the above embodiments, this description of the preferred embodiments and methods is not meant to be construed in a limiting sense. For example, the term Java in the specification or claims should be construed to cover successor programming languages or other programming languages using basic Java concepts (the use of generic instructions, such as bytecodes, to indicate the operation of a virtual machine). It should also be understood that all aspects of the present invention are not to be limited to the specific descriptions, or to configurations set forth herein. Some modifications in form and detail the various embodiments of the disclosed invention, as well as other variations in the present invention, will be apparent to a person skilled in the art upon reference to the present disclosure. It is therefore contemplated that the following claims will cover any such modifications or variations of the described embodiment as falling within the true spirit and scope of the present invention.

CLAIMS

1. A system comprising:

a central processing unit; and

a hardware accelerator operably connected to the central processing unit, the hardware accelerator adapted to translate stack-based instructions into register-based instructions native to the central processing unit.

2. The system of Claim 1, wherein the stack-based instructions are associated with a virtual machine.

3. The system of Claim 1, wherein the stack-based instructions are Java bytecode.

4. The system of Claim 1, wherein the hardware accelerator implements at least part of a Java virtual machine.

5. The system of Claim 1, wherein the hardware accelerator is connected between a memory and the central processing unit.

6. The system of Claim 5, wherein the hardware accelerator is connected between an instruction cache and the central processing unit.

7. The system of Claim 1, wherein the hardware accelerator is adapted to manage a java stack.

8. The system of Claim 1, wherein the hardware accelerator is adapted to store at least some of a Java operand stack in a register file connected to the central processing unit.

9. The system of Claim 8, wherein the hardware accelerator has access to the data bus of the central processing unit.

10. The system of Claim 8, wherein the hardware accelerator is adapted to swap parts of the operand stack are in and out of the register file from a memory.

11. The system of Claim 8, wherein the central processing unit is operably connected to a native register file and a register file controlled by the hardware accelerator

12. The system of Claim 11, wherein the at least some of the Java operand stack is stored in the register file controlled by the hardware accelerator.

13. The system of Claim 8, wherein the central processing unit is operably connected to a native register file and wherein the at least some of the Java operand stack is stored in the native register file.

14. The system of Claim 8, wherein the hardware controller is further adapted to store at least some variables in the register file.

15. The system of Claim 8, wherein the hardware accelerator is incorporated within the central processing unit.

16. The system of Claim 1, wherein the hardware accelerator has access to at least one bus of the central processing unit.

17. The system of Claim 1, wherein the hardware accelerator is adapted to examine the stack-based instructions to determine whether multiple stack-based instructions can be combined into fewer register-based instructions.

18. The system of Claim 17, wherein multiple stack-based instructions pass through the hardware accelerator concurrently to allow for the operation of the combining logic.

19. The system of Claim 1, wherein the hardware accelerator is divided into pipelined stages.

20. The system of Claim 1, wherein the hardware accelerator is adapted to be flushed under predetermined conditions.

21. The system of Claim 1, wherein the central processing unit and hardware accelerator are on the same chip.

22. The system of Claim 1, wherein the hardware accelerator produces an exception upon at least one of the stack-based instructions, and wherein the central processing unit will, in software, translate the at least one of the stack-based instructions causing the exception.

23. The system of Claim 1, wherein the hardware accelerator is incorporated within the central processing unit.

24. A system comprising:

a central processing unit; and

a hardware java accelerator operably connected to the central processing unit, the hardware java accelerator adapted to translate java bytecodes into instructions native to the central processing unit.

25. A system comprising:

a central processing unit: and

a hardware accelerator operably connected to the central processing unit, the hardware accelerator adapted implement at least part of a virtual machine associated with a computer language, the hardware accelerator adapted to translate instructions for the virtual machine into native instructions for the central processing unit.

26 . A method comprising:

moving a stack-based instruction from memory to a hardware accelerator;

in the hardware accelerator, converting the stack-based instruction into a register-based instruction native to a central processing unit; and

in the central processing unit, executing the register-based instruction.

27. The method of Claim 26, wherein the stack-based instructions are associated with a virtual machine.

28. The method of Claim 26, wherein the stack-based instructions are Java bytecode.

29. The method of Claim 26, wherein the accelerator implements at least part of a Java virtual machine.

30. The method of Claim 26, further comprising, in the hardware accelerator, managing a java stack.

31. The method of Claim 26, further comprising storing at least some of a Java operand stack in a register file connected to the central processing unit.

32. The method of Claim 26, wherein parts of the operand stack are swapped in and out of the register file from a memory by the hardware accelerator.

33. The method of Claim 26, wherein the hardware accelerator examines the stack-based instructions to determine whether multiple stack-based instructions can be combined into fewer register-based instructions.

34. The method of Claim 26, further comprising producing an exception in the hardware accelerator upon at least one stack-based instruction, and translating the at least one stack-based instruction causing the exception in software in the central processing unit.

35. The method of Claim 26, wherein the central processing unit and hardware accelerator are on the same chip.

36. The method of Claim 26, wherein the hardware accelerator is incorporated within the central processing unit.

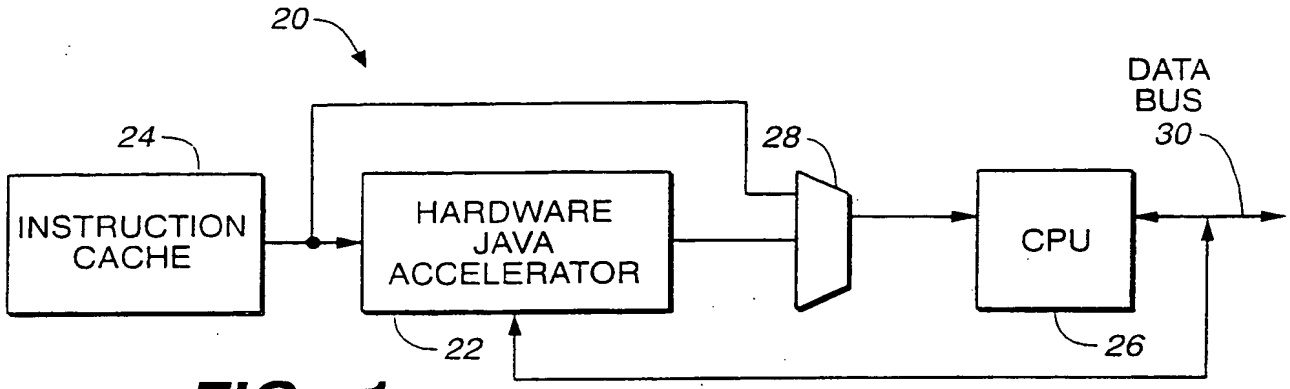


FIG._1

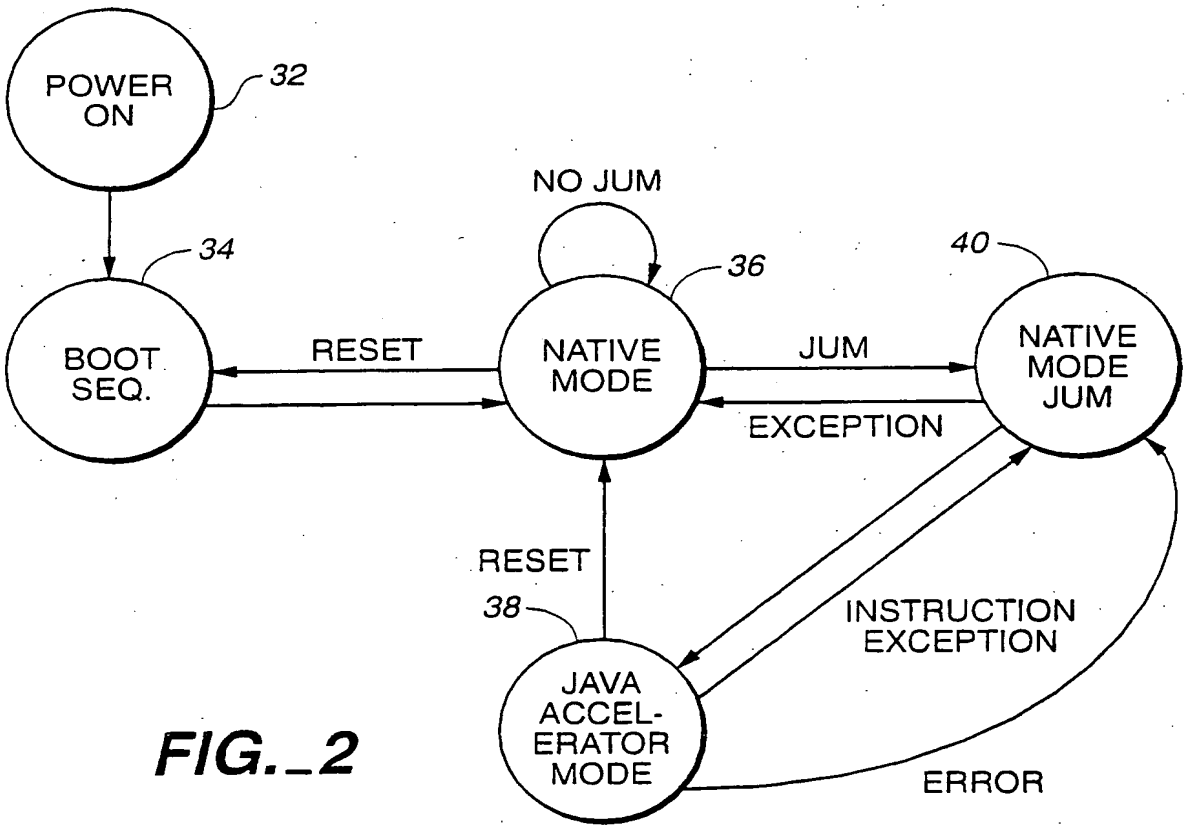


FIG._2

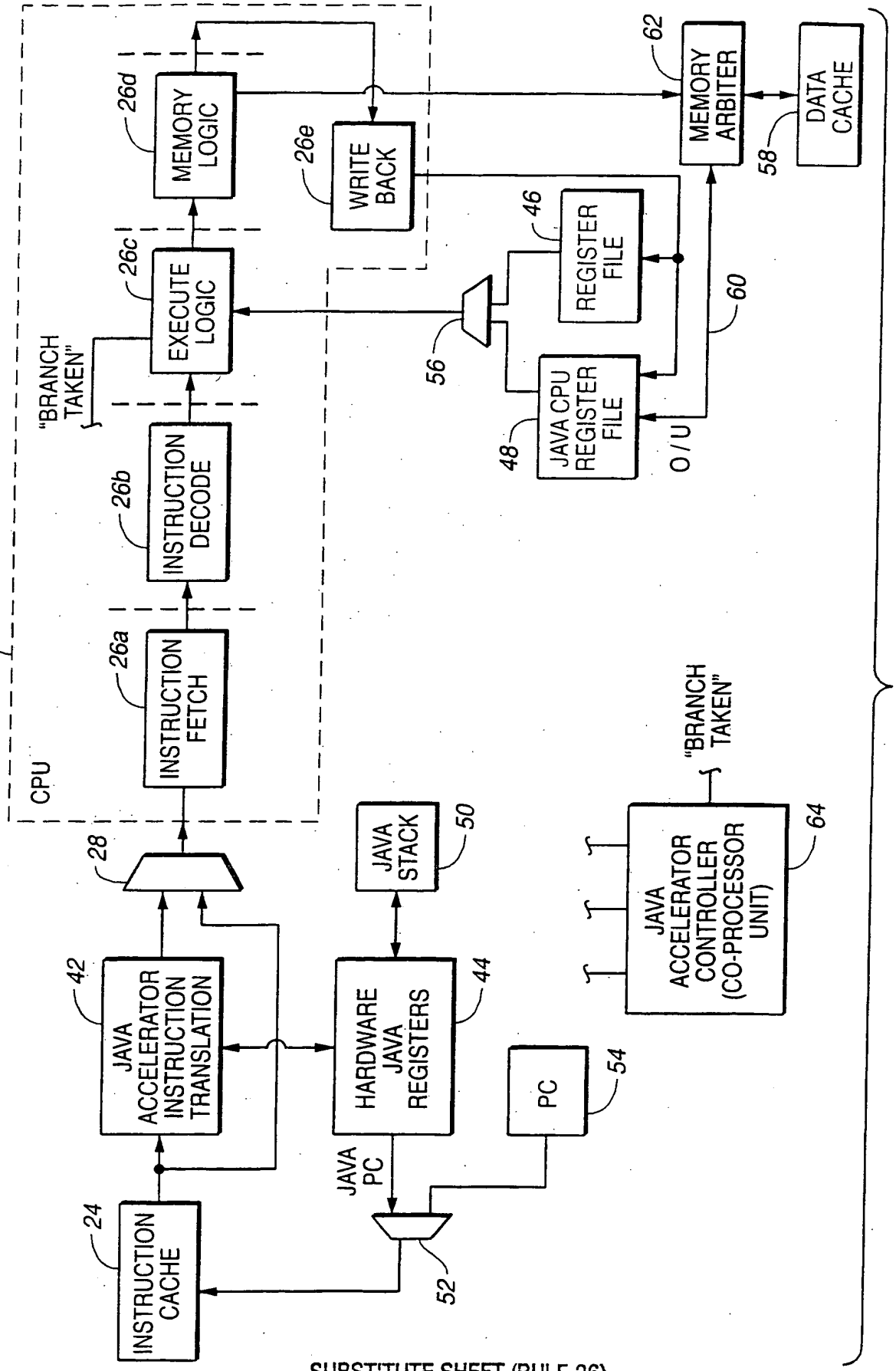


FIG._3

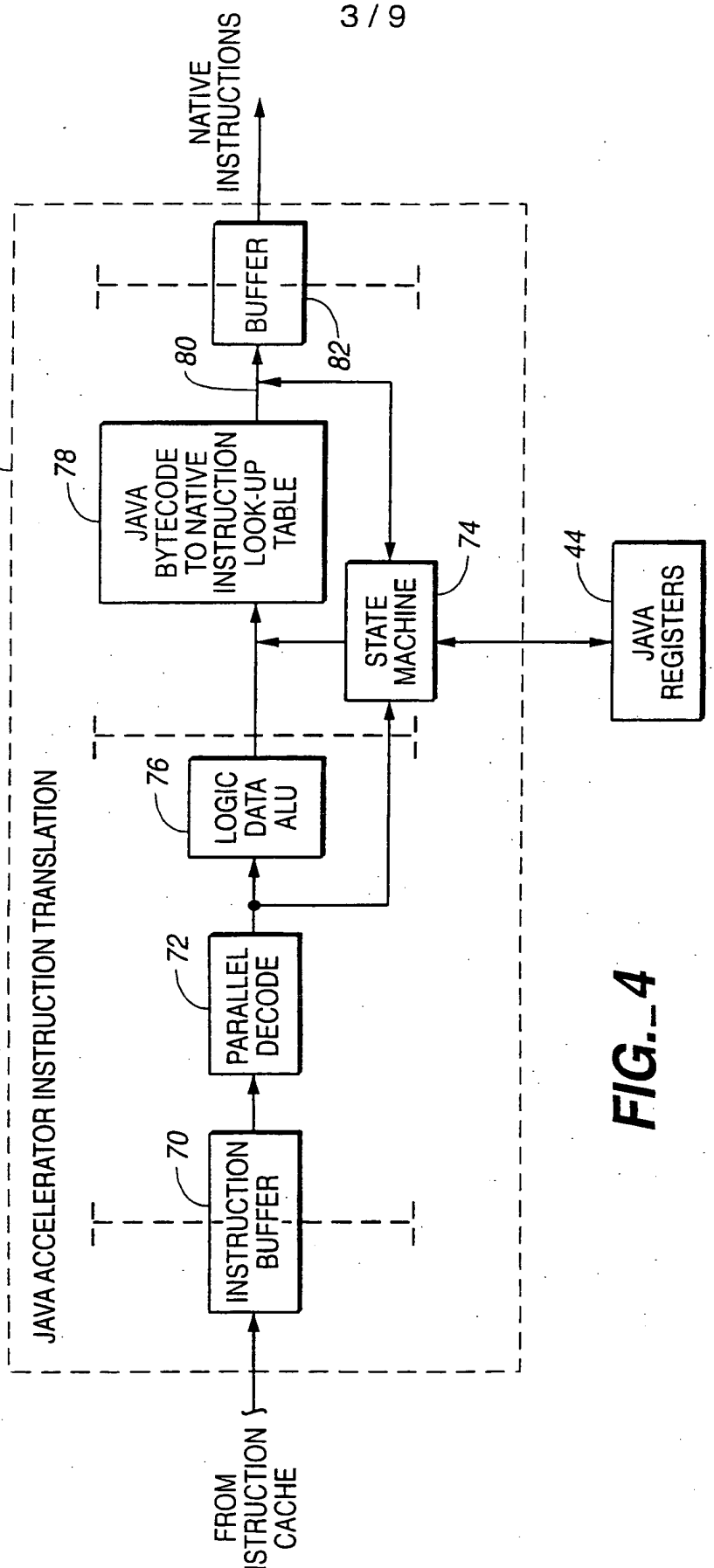

NATIVE INSTRUCTIONS
BUFFER
82
80
42
78
JAVA BYTECODE TO NATIVE INSTRUCTION LOOK-UP TABLE
74
STATE MACHINE
44
JAVA REGISTERS
JAVA ACCELERATOR INSTRUCTION TRANSLATION
76
LOGIC DATA ALU
72
PARALLEL DECODE
70
INSTRUCTION BUFFER
FROM INSTRUCTION CACHE

FIG._4

I. INSTRUCTION TRANSLATION

JAVA BYTECODE		NATIVE INSTRUCTION
iadd	⇨	ADD R1, R2

II. JAVA REGISTER

PC = VALUE A OPTOP = VALUE B (R1) VAR = VALUE C	⇨	PC = VALUE A + 1 OPTOP = VALUE B - 1 (R2) VAR = VALUE C

III. JAVA CPU REGISTER FILE

	R0	0001			R0 0001
CONTAINS VALUE OF TOP OF OPERAND STACK →	R1	0150	⇨	NOT A VALID STACK VALUE →	R1 0150
	R2	1210		CONTAINS VALUE OF THE TOP OF OPERAND STACK →	R2 1360
	R3	0007			R3 0007
	R4	0005			R4 0005
	R5	0006			R5 0006
CONTAINS FIRST VARIABLE →	R6	1221			R6 1221
	R7	1361			R7 1361

IV. MEMORY

OPTOP = VALUE B →	-	0150			- 0150
(VALUE B - 1)	-	1210	⇨	OPTOP = VALUE B - 1	- 1360
	-	0007			- 0007
	-	0005			- 0005
	-	0006			- 0006
	-	0001			- 0001
	-	4427			- 4427
VAR = VALUE C	-	1221			- 1221
	-	1361			- 1361
	-	1101			- 1101

FIG._5

I. INSTRUCTION TRANSLATION

JAVA BYTECODE

iload_n
iadd

NATIVE INSTRUCTION

ADD R6, R1

II. JAVA REGISTER

PC = VALUE A
OPTOP = VALUE B (R1)
VAR = VALUE C

PC = VALUE A + 2
OPTOP = VALUE B (R1)
VAR = VALUE C

III. JAVA CPU REGISTER FILE

	R0	0001		R0	0001
CONTAINS VALUE OF TOP OF OPERAND STACK →	R1	0150	CONTAINS VALUE OF TOP OF STACK →	R1	1371
	R2	1210		R2	1210
	R3	0007		R3	0007
	R4	0005		R4	0005
	R5	0006		R5	0006
CONTAINS FIRST VARIABLE →	R6	1221	CONTAINS FIRST VARIABLE →	R6	1221
	R7	1361		R7	1361

IV. MEMORY

OPTOP = VALUE B →	-	0150
	-	1210
	-	0007
	-	0005
	-	0006
	-	0001
	-	4427
VAR = VALUE C	-	1221
	-	1361
	-	1101

OPTOP = VALUE B	-	1371
	-	1210
	-	0007
	-	0005
	-	0006
	-	0001
	-	4427
VAR = VALUE C	-	1221
	-	1361
	-	1101

FIG._6

Opcodes Mnemonic	Opcode xHH	Excep Gen
nop	0x00	
aconst_null	x01	
iconst_m1	x02	
iconst_n(0-5)	x03 - x08	
lconst_n(0-1)	x09 - x0a	
fconst_n(0-2)	x0c - x0d	
dconst_n(0-1)	x0e -x0f	
bipush	x10	
sipush	x11	
ldc	x12	y
ldc_w	x13	y
ldc2_w	x14	y
iload	x15	
lload	x16	
fload	x17	
dload	x18	
aload	x19	
iload_n(0-3)	x1a - x1d	
lload_n(0-3)	x1e - x21	
fload_n(0-3)	x22 - x25	
dload_n(0-3)	x26 - x29	
aload_n(0-3)	x2a - x2d	
iaload	x2e	
laload	x2f	
faload	x30	
daload	x31	
aaload	x32	
baload	x33	
caload	x34	
saload	x35	
istore	x36	
lstore	x37	
fstore	x38	
dstroe	x39	
astroe	x3a	
istore_n(0-3)	x3b - x3e	
lstore_n(0-3)	x3f - x42	
fstore_n(0-3)	x43 - x46	
dstore_n(0-3)	x47 - x4a	
astore_n(0-3)	x4b - x4e	
iastore	x4f	
lastore	x50	
fastroe	x51	
dastore	x52	
bastore	x53	
aastore	x54	
castroe	x55	
sastore	x56	

FIG._7A

pop	x57	
pop2	x58	
dup	x59	
dup_x1	x5a	
dup_x2	x5b	
dup2	x5c	
dup2_x1	x5d	
dup2_x2	x5e	
swap	x5f	
iadd	x60	
ladd	x61	
fadd	x62	y
dadd	x63	y
isub	x64	
lsub	x65	
fsub	x66	y
dsub	x67	y
imul	x68	
lmul	x69	
fmul	x6a	y
dmul	x6b	y
idiv	x6c	y
ldiv	x6d	y
fdiv	x6e	y
ddiv	x6f	y
irem	x70	y
lrem	x71	y
frem	x72	y
drem	x73	y
ineg	x74	
lneg	x75	
fneg	x76	y
dneg	x77	y
ishl	x78	
lshl	x79	
ishr	x7a	
lshr	x7b	
iushr	x7c	
lushr	x7d	
iand	x7e	
land	x7f	
ior	x80	
lor	x81	
ixor	x82	
lxor	x83	
iinc	x84	
i2l	x85	y
i2f	x86	y
i2d	x87	y
l2i	x88	y
l2f	x89	y
l2d	x8a	y

FIG._7B

f2i	x8b	y
f2l	x8c	y
f2d	x8d	y
d2i	x8e	y
d2l	x8f	y
d2f	x90	y
i2b	x91	
i2c	x92	
i2s	x93	
lcmp	x94	y
fcmpl	x95	y
fcmpg	x96	y
dcmpl	x97	y
dcmpg	x98	y
ifeq	x99	
ifne	x9a	
iflt	x9b	
ifge	x9c	
ifgt	x9d	
ifle	x9e	
if_icmpeq	x9f	
if_icmpne	xa0	
if_icmplt	xa1	
if_acmpge	xa2	
if_cmpgt	xa3	
if_icmple	xa4	
if_acmpeq	xa5	
if_acmpne	xa6	
goto	xa7	
jsr	xa8	
ret	xa9	
tableswitch	xaa	y
lookupswitch	xab	y
ireturn	xac	
lreturn	xad	
freturn	xae	
dreturn	xaf	
areturn	xb0	
return	xb1	
getstatic	xb2	y
putstatic	xb3	y
getfield	xb4	y
putfield	xb5	y
invokevirtual	xb6	y
invokespecial	xb7	y
invokestatic	xb8	y
invokeinterface	xb9	y
xxunsedxxx	xba	y
new	xbb	y
newarray	xbc	y
anewarray	xbd	y
arraylength	xbe	y

FIG._7C

athrow	xbf	y
checkcast	xco	y
instanceof	xc1	y
monitorenter	xc2	y
monitorexit	xc3	y
wide	xc4	y
multianewarray	xc5	y
ifnull	xc6	y
ifnonnull	xc7	y
goto_w	xc8	
jsr_w	xc9	
ldc_quick	xcb	y
ldc_w_quick	xcc	y
ldc2_w_quick	xcd	y
getfield_quick	xce	y
putfield_quick	xcf	y
getfield2_quick	xd0	y
putfield2_quick	xd1	y
getstatic_quick	xd2	y
putstatic_quick	xd3	y
gtestatic2_quick	xd4	y
putstatic2_quick	xd5	y
invokevirtual_quick	xd6	y
invokenonvirtual_quick	xd7	y
invokesuper_quick	xd8	y
invokestatic_quick	xd9	y
invokeinterface_quick	xda	y
invokevirtualobject_quick	xdb	y
new_quick	xdc	y
anewarray_quick	xde	y
multinewarray_quick	xdf	y
checkcast_quick	xe0	y
instanceof_quick	xe1	y
invokevirtual_quick_w	xe2	y
getfield_quick_w	xe3	y
putfield_quick_w	xe4	y
breakpoint	xca	y
impdep1	xfe	y
impdep2	xff	y

FIG._7D

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/28782

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 9/45, 9/445
US CL : 717/5
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 717/5, 395/800, 395/570,

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

STN, WEST, EAST, IEEE

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	CACAO- a 64 bit Java VM, just-in-time compiler, November 1997, vol 9, Krall et al, page 1017-1030.	1-36
A, P	US 5,898,885 A (DICKOL et al) 27 April 1999, see entire document	1-36
A, P	US 5,898,850 A (DICKOL et al) 27 April 1999, see entire document	1-36
A, P	US 5,875,336 A (DECKOL et al) 23 February 1999, see entire document	1-36
A	Efficient Java VM Just-in-Time Compilation, IEEE, 1998, see entire document	1-36

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 21 MARCH 2000	Date of mailing of the international search report 27 JUN 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-0040	Authorized officer TARIQ HAFIZ Telephone No. (703) 305-9643

Form PCT/ISA/210 (second sheet) (July 1998) ★

INTERNATIONAL SEARCH REPORT

Int[illegible]onal application No.
PCT/US99/28782

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	SGI Webforce 02 is a one-step Web authoring platform, InfoWorld, 20 January 1997, see entire document	1-36

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/28782

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☒ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.